Patent Application
Docket No.:47242-0028USPT
CUSTOMER NUMBER 23932

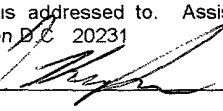
APPLICATION FOR UNITED STATES LETTERS PATENT

for

PERSONAL VOICE-BASED INFORMATION RETRIEVAL SYSTEM

by

Alexander Kurganov

EXPRESS MAIL MAILING LABEL
EXPRESS MAIL NO.. EK506616987US DATE OF DEPOSIT: February 5, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 C.F.R. 1.10 on the date indicated above and is addressed to. Assistant Commissioner for Patents, Attn: Box Patent Application, Washington D.C. 20231
Signature: 

PERSONAL VOICE-BASED INFORMATION RETRIEVAL SYSTEM

CROSS-REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of priority from United States Provisional Application Serial No. 60/180,343, filed February 4, 2000 entitled "Personal Voice-Based Information Retrieval System"

FIELD OF THE INVENTION

The present invention relates generally to the field of providing information access. In particular, the invention relates to a personalized system for accessing information from the Internet or other information sources using speech commands.

BACKGROUND OF THE INVENTION

Popular methods of information access and retrieval using the Internet or other computer networks can be time-consuming and complicated. A user must frequently wade through vast amounts of information provided by an information source or web site in order obtain a small amount of relevant information. This can be time-consuming, frustrating, and, depending on the access method, costly. A user is required to continuously identify reliable sources of information and, if these information sources are used frequently, repeatedly access these sources.

Current methods of accessing information stored on computer networks, such as Wide Area Networks (WANs), Local Area Network (LANs) or the Internet, require a user to have access to a computer. While computers are becoming increasingly smaller and easier to transport, using a computer to access information is still more difficult than simply using a telephone. Since speech recognition systems allow a user to convert his voice into a computer-usable message, telephone access to digital information is becoming more and more feasible. Voice recognition technology is growing in its ability to allow users to use a wide vocabulary. Further, such technology is quite accurate when a single, known user only needs to use a small vocabulary.

Therefore, a need exists for an information access and retrieval system and method that allows users to access frequently needed information from information sources on networks by using a telephone and simple speech commands.

SUMMARY OF THE INVENTION

One object of the preferred embodiment of the present invention is to allow users to customize a voice browsing system.

A further object of the preferred embodiment is to allow users to customize the information retrieved from the Internet or other computer networks and accessed by speech commands over telephones.

Another object of the preferred embodiment is to provide a secure and reliable retrieval of information over the Internet or other computer networks using predefined verbal commands assigned by a user.

The present invention provides a solution to these and other problems by providing a new system for retrieving information from a network such as the Internet. A user creates a user-defined record in a database that identifies an information source, such as a web site, containing information of interest to the user This record identifies the location of the information source and also contains a recognition grammar assigned by the user. Upon receiving a speech command from the user that is described in the assigned recognition grammar, a network interface system accesses the information source and retrieves the information requested by the user.

In accordance with the preferred embodiment of the present invention, a customized, voice-activated information access system is provided. A user creates a descriptor file defining specific information found on a web site the user would like to access in the future. The user then assigns a pronounceable name or identifier to the selected content and this pronounceable name is saved in a user-defined database record as a recognition grammar along with the URL of the selected web site.

In the preferred embodiment, when a user wishes to retrieve the previously defined web-based information, a telephone call is placed to a media server. The user provides speech commands to the media server that are described in the recognition grammar assigned to the desired search. Based upon the recognition grammar, the media server retrieves the user-defined record from a database and passes the information to a web

browsing server which retrieves the information from associated web site. The retrieved information is then transmitted to the user using a speech synthesis software engine.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 displays a personal information selection system used with the preferred embodiment of the present invention;

FIG. 2 displays a web page displayed by the clipping client of the preferred embodiment,

FIG. 3 is a block diagram of a voice browsing system used with preferred embodiment of the present invention;

FIG 4 is a block diagram of a user-defined database record created by preferred embodiment of the present invention;

FIG 5 is a block diagram of a media server used by the preferred embodiment, and

FIG. 6 is a block diagram of a web browsing server used by the preferred embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention uses various forms of signal and data transmission to allow a user to retrieve customized information from a network using speech communication. In the preferred embodiment of the present invention, a user associates information of interest found on a specific information source, such as a web site, with a pronounceable name or identification word This pronounceable name/identification word forms a recognition grammar in the preferred embodiment. When the user wishes to retrieve the selected information, he may use a telephone or other voice enabled device to access a voice browser system. The user then speaks a command described in the recognition grammar associated with the desired information. The voice browsing system then accesses the associated information source and returns to the user, using a voice synthesizer, the requested information.

Referring to FIG. 1, a user 100 uses a computer 102 to access a network, such as a WAN, LAN, or the Internet, containing various information sources. In the preferred embodiment, the user 100 access the Internet 104 and begins searching for web sites 106,

which are information sources that contain information of interest to the user. When the user 100 identifies a web site 106 containing information the user would like to access using only a voice enabled device, such as a telephone, and the voice browsing system 108, the user initiates a "clipping client" engine 110 on his computer 102.

The clipping client 110 allows a user 100 to create a set of instructions for use by the voice browsing system 108 in order to report personalized information back to the user upon request. The instruction set is created by "clipping" information from the identified web site A user 100 may be interested in weather for a specific city, such as Chicago. The user 100 identifies a web site from which he would like to obtain the latest Chicago weather information. The clipping client 110 is then activated by the user 100.

The clipping client 110 displays the selected web site in the same manner as a conventional web browser such as Microsoft's® Internet Explorer. FIG. 2 depicts a sample of a web page 200 displayed by the clipping client 110. The user 100 begins creation of the instruction set for retrieving information from the identified web site by selecting the uniform resource locator (URL) address 202 for the web site (i.e., the web site address). In the preferred embodiment, this selection is done by highlighting and copying the URL address 202. Next, the user selects the information from the displayed web page that he would like to have retrieved when a request is made. Referring to FIG. 2, the user would select the information regarding the weather conditions in Chicago 204. The web page 200 may also contain additional information such as advertisements 206 or links to other web sites 208 which are not of interest to the user. The clipping client 110 allows the user to select only that portion of the web page containing information of interest to the user. Therefore, unless the advertisements 206 and links 208 displayed on the web page are of interest to the user, he would not select this information. Based on the web page information 204 selected by the user, the clipping client 110 creates a content descriptor file containing a description of the content of the selected web page. This content descriptor file indicates where the information selected by the user is located on the web page. In the preferred embodiment, the content descriptor file is stored within the web browsing server 302 shown in FIG. 3. The web browsing server 302 will be discussed below.

Table 1 below is an example of a content descriptor file created by the clipping client of the preferred embodiment. This content descriptor file relates to obtaining weather information from the web site www.cnn.com.

TABLE 1

```
table name : portalServices
column :
 service
content:
   weather
column:
 config
content:
 [cnn]
 Input=_zip

 URL=http://cgi.cnn.com/cgi-bin/weather/redirect?zip=_zip

 Pre-filter="\n" "
 Pre-filter="<[^<>]+>""
 Pre-filter=/\s+/ /
 Pre-filter="[\(\)\|]"!"

 Output=_location
 Output=first_day_name
 Output=first_day_weather
 Output=first_day_high_F
 Output=first_day_high_C
 Output=first_day_low_F
 Output=first_day_low_C
 Output=second_day_name
 Output=second_day_weather
 Output=second_day_high_F
 Output=second_day_high_C
 Output=second_day_low_F
 Output=second_day_low_C
 Output=third_day_name
 Output=third_day_weather
 Output=third_day_high_F
 Output=third_day_high_C
 Output=third_day_low_F
 Output=third_day_low_C
 Output=fourth_day_name
 Output=fourth_day_weather
 Output=fourth_day_high_F
 Output=fourth_day_high_C
 Output=fourth_day_low_F
 Output=fourth_day_low_C
 Output=undef
 Output=_current_time
 Output=_current_month
 Output=_current_day
 Output=_current_weather
```

```
Output=_current_temperature_F
Output=_current_temperature_C
Output=_humidity
Output=_wind
Output=_pressure
Output=_sunrise
Output=_sunset

Regular_expression=WEB SERVICES: (.+) Forecast FOUR-DAY
FORECAST (\S+)
(\S+) HI
GH (\S+) F (\S+) C LOW (\S+) F (\S+) C (\S+) (\S+) HIGH
(\S+) F (\S+) C LOW
(\S+
) F (\S+) C (\S+) (\S+) HIGH (\S+) F (\S+) C LOW (\S+) F
(\S+) C (\S+) (\S+)
HIG
H (\S+) F (\S+) C LOW (\S+) F (\S+) C WEATHER MAPS RADAR
(.+) Forecast
CURRENT C
ONDITIONS (.+) !local!, (\S+) (\S+) (.+) Temp: (\S+) F,
(\S+) C Rel.
Humidity: (
\S+) Wind: (.+) Pressure: (.+) Sunrise: (.+) Sunset: (.+)
```

Finally, the clipping client 110 prompts the user to enter an identification word or phrase that will be associated with the identified web site and information. For example, the user could associate the phrase "Chicago weather" with the selected URL 202 and related weather information 204. The identification word or phrase is stored as a personal recognition grammar that can now be recognized by a speech recognition engine of the voice browsing system 108 which will be discussed below. The personal recognition grammar, URL address 202, and a command for executing a content extraction agent are stored within a database used by the voice browser system 108 which will be discussed below.

The voice browsing system 108 used with the preferred embodiment will now be described in relation to FIG. 3. A database 300 designed by Webley Systems Incorporated is connected to one or more web browsing servers 302 as well as to one or more media servers 304. The database may store information on magnetic media, such as a hard disk drive, or it may store information via other widely acceptable methods for storing data, such as optical disks. The media servers 304 function as user interface systems that provide access to the voice browsing system 108 from a user's voice

enabled device 306 (i.e , any type of wireline or wireless telephone, Internet Protocol (IP) phones, or other special wireless units). The database 300 contains a section that stores the personal recognition grammars and related web site information generated by the clipping client 110. A separate record exists for each web site defined by the user. An example of a user – defined web site record is shown in FIG 4. Each user-defined web site record 400 contains the recognition grammar 402 assigned by the user, the associated Uniform Resource Locator (URL) 404, and a command that enables the "content extraction agent" 406 and retrieves the appropriate content descriptor file required to generate proper requests to the web site and to properly format received data. The web-site record 400 also contains the timestamp 408 indicating the last time the web site was accessed. The content exaction agent is described in more detail below.

The database 300 may also contain a listing of pre-recorded audio files used to create concatenated phrases and sentences. Further, database 300 may contain customer profile information, system activity reports, and any other data or software servers necessary for the testing or administration of the voice browsing system 108.

The operation of the media servers 304 will now be discussed in relation to FIG. 5. The media servers 304 function as user interface systems since they allow a user to access the voice browsing system 108 via a voice enabled device 306. In the preferred embodiment, the media servers 304 contain a speech recognition engine 500, a speech synthesis engine 502, an Interactive Voice Response (IVR) application 504, a call processing system 506, and telephony and voice hardware 508 that is required to enable the voice browsing system 108 to communicate with the Public Switched Telephone Network (PSTN) 308. In the preferred embodiment, each media server is based upon Intel's Dual Pentium III 730 MHz microprocessor system.

The speech recognition function is performed by a speech recognition engine 500 that converts voice commands received from the user's voice enabled device 10 (i.e., any type of wireline or wireless telephone, Internet Protocol (IP) phones, or other special wireless units) into data messages. In the preferred embodiment, voice commands and audio messages are transmitted using the PSTN 308 and data is transmitted using the TCP/IP communications protocol. However, one skilled in the art would recognize that other transmission protocols may be used. Other possible transmission protocols would include SIP/VoIP (Session Initiation Protocol/Voice over IP), Asynchronous Transfer Mode (ATM) and Frame Relay. A preferred speech recognition engine is developed by Nuance Communications of 1380 Willow Road, Menlo Park, California 94025

(www.nuance.com) The Nuance engine capacity is measured in recognition units based on CPU type as defined in the vendor specification. The natural speech recognition grammars (i e., what a user can say that will be recognized by the speech recognition engine) were developed by Webley Systems.

In the preferred embodiment, when a user access the voice browsing system 108, he will be prompted if he would like to use his "user-defined searches " If the user answers affirmatively, the media servers 304 will retrieve from the database 300 the personal recognition grammars 402 defined by the user while using the clipping client 110.

The media servers 304 also contain a speech synthesis engine 502 that converts the data retrieved by the web browsing servers 302 into audio messages that are transmitted to the user's voice enabled device 306 A preferred speech synthesis engine is developed by Lernout and Hauspie Speech Products, 52 Third Avenue, Burlington, Massachusetts 01803 (www.lhsl.com)

A further description of the web browsing server 302 will be provided in relation to FIG. 6. The web browsing servers 302 provide access to data stored on any computer network including the Internet 104, WANs or LANs. The web browsing servers 302 receive responses from web sites 106 and extract the data requested by the user. This task is known as "content extraction." The web browsing server 302 is comprised of a content extraction agent 600, a content fetcher 602, and the content descriptor file 604. Each of these are software applications and will be discussed below.

Upon receiving a user-defined web site record 400 from the database 300 in response to a user request, the web browsing server 302 invokes the "content extraction agent" command 406 contained in the record 400. The content extraction agent 600 retrieves the content descriptor file 604 associated with the user-defined record 400. As mentioned, the content descriptor file 604 directs the extraction agent where to extract data from the accessed web page and how to format a response to the user utilizing that data. For example, the content descriptor file 604 for a web page providing weather information would indicate where to insert the "city" name or ZIP code in order to retrieve Chicago weather information. Additionally, the content descriptor file 604 for each supported URL indicates the location on the web page where the response information is provided The extraction agent 600 uses this information to properly extract from the web page the information requested by the user.

The content extraction agent 600 can also parse the content of a web page in which the user-desired information has changed location or format. This is accomplished based on the characteristic that most hypertext documents include named objects like tables, buttons, and forms that contain textual content of interest to a user When changes to a web page occur, a named object may be moved within a document, but it still exists. Therefore, the content extraction agent 600 simply searches for the relevant name of desired object. In this way, the information requested by the user may still be found and reported regardless of changes that have occurred.

Table 2 below contains source code for a content extraction agent 600 used by the preferred embodiment.

TABLE 2

```
#!/usr/local/www/bin/sybperl5
#$Header:
/usr/local/cvsroot/webley/agents/service/web_dispatch.pl,v
1.6
# Dispatches all web requests

#http://wcorp.itn.net/cgi/flstat?carrier=ua&flight_no=155&m
on_abbr=jul&date=
6&stamp=OhLN~PdbuuE*itn/ord,itn/cb/sprint_hd
#http://cgi.cnnfn.com/flightview/rlm?airline=amt&number=300


require "config_tmp.pl";

# check parameters
die "Usage: $0 service [params]\n" if $#ARGV < 1;
#print STDERR @ARGV;

# get parameters
my ( $service, @param ) = @ARGV;

# check service
my %Services = (
                    weather_cnn => 'webget.pl weather_cnn',
                    weather_lycos => 'webget.pl
weather_lycos',
                    weather_weather => 'webget.pl
weather_weather',
                    weather_snap => 'webget.pl
weather_snap',
                    weather_infospace => 'webget.pl
```

```
    weather_infospace',
                        stockQuote_yahoo => 'webget.pl stock',
                        flightStatus_itn => 'webget.pl
    flight_delay',
                        yellowPages_yahoo => 'yp_data.pl',
                        yellowPages_yahoo => 'yp_data.pl',
                        newsHeaders_newsreal => 'news.pl',
                        newsArticle_newsreal => 'news.pl',
                        );
    # test param
    my $date = `date`;
    chop( $date );
    my ( $short_date ) = $date =~ /\s+(\w{3}\s+\d{1,2})\s+/;
    my %Test = (
                        weather_cnn => '60053',
                        weather_lycos => '60053',
                        weather_weather => '60053',
                        weather_snap => '60053',
                        weather_infospace => '60053',
                        stockQuote_yahoo => 'msft',
                        flightStatus_itn => 'ua 155 ' .
    $short_date,
                        yellowPages_yahoo => 'tires 60015',
                        newsHeaders_newsreal => '1',
                        newsArticle_newsreal => '1 1',
                        );
    die "$date: $0: error: no such service: $service (check
    this script)\n"
    unless $Services{ $service };

    # prepare absolute path to run other scripts
    my ( $path, $script ) = $0 =~ m|^(.*/)([^/]*)|;

    # store the service to compare against datatable
    my $service_stored = $service;

    # run service
    while( !( $response = `$path$Services{ $service } @param` )
    ) {
         # response failed
         # check with test parameters
         $response = `$path$Services{ $service } $Test{
    $service }`;
    #    print "test: $path$Services{ $service } $Test{
    $service }";
         if ( $response ) {
              $service = &switch_service( $service );
    #         print "Wrong parameter values were supplied:
    $service -
    @param\n";
```

```
#          die "$date: $0: error: wrong parameters: $service
-
@param\n";
     }
     else {
          # change priority and notify
          $service = &increase_attempt( $service );
     }
}

# output the response
print $response;

sub increase_attempt {
     my ( $service ) = @_;
     my ( $service_name ) = split( /_/, $service );
     print STDERR "$date: $0: attn: changing priority for
service:
$service\n";
     # update priority
     &db_query( "update mcServiceRoute "
                    . "set priority = ( select max( priority
) from
mcServiceRoute "
                    . "where service = '$service_name' ) + 1,
"
                    . "date = getdate(), "
                    . "attempt = attempt + 1 "
                    . "where route = '$script $service'" );
#    print "---$route===\n";
     # find new route
     my $route = @{ &db_query( "select route from
mcServiceRoute "
                                        . "where service =
'$service_name' "
                                        . "and attempt < 5
"
                                        . "order by
priority ")
                          } -> [ 0 ]{ route };
     &db_query( "update mcServiceRoute "
                    . "set attempt = 0 "
                    . "where route = '$script $service'" )
          if ( $route eq "$script $service"
                or $route eq "$script $service_stored" );
     ( $service_name, $service ) = split( /\s+/, $route );
     die "$date: $0: error: no route for the service:
$service (add
more)\n"
          unless $service;
     return $service;
```

```
}
sub switch_service {
     my ( $service ) = @_;
     my ( $service_name ) = split( /_/, $service );
     print STDERR "$date: $0: attn: changing priority for
service:
$service\n";
     # update priority
     &db_query( "update mcServiceRoute "
                    . "set priority = ( select max( priority
) from
mcServiceRoute "
                    . "where service = '$service_name' ) + 1,
"
                    . "date = getdate() "
                    . "where route = '$script $service'" );
#    print "---$route===\n";
     # find new route
     my $route = @{ &db_query( "select route from
mcServiceRoute "
                                    . "where service =
'$service_name' "
                                    . "and attempt < 5
"
                                    . "order by
priority ")
                         } -> [ 0 ]{ route };
     die "$date: $0: error: there is the only service:
$route (add
more)\n"
          if ( $route eq "$script $service"
               or $route eq "$script $service_stored" );
     ( $service_name, $service ) = split( /\s+/, $route );
     die "$date: $0: error: no route for the service:
$service (add
more)\n"
          unless $service;
     return $service;
}
```

Table 3 below contains source code of the content fetcher 602 used with the content extraction agent 600 to retrieve information from a web site.

TABLE 3

```
#!/usr/local/www/bin/sybperl5
```

```
#-T
# -w
# $Header:
/usr/local/cvsroot/webley/agents/service/webget.pl,v 1.4
# Agent to get info from the web.
# Parameters: service_name [service_parameters], i.e. stock
msft or weather
60645
# Configuration stored in files service_name.ini
# if this file is absent the configuration is received from
mcServices table
# This script provides autoupdate to datatable if the .ini
file is newer.

$debug = 1;
use URI::URL;
use LWP::UserAgent;
use HTTP::Request::Common;
use Vail::VarList;
use Sybase::CTlib;
use HTTP::Cookies;
#print "Sybase::CTlib $DB_USR, $DB_PWD, $DB_SRV;";
open( STDERR, ">>$0.log" ) if $debug;
#open( STDERR, ">&STDOUT" );
$log = `date`;
#$response = `./url.pl
"http://cgi.cnn.com/cgi-bin/weather/redirect?zip=60605"`;
#$response = `pwd`;
#print STDERR "pwd = $response\n";
#$response = `ls`;
#print STDERR "ls = $response\n";
chop( $log );
$log .= "pwd=" . `pwd`;
chop( $log );

#$debug2 = 1;

my $service = shift;
$log .= " $service: ". join( ':', @ARGV ) . "\n";
print STDERR $log if $debug;
#$response = `./url.pl
"http://cgi.cnn.com/cgi-bin/weather/redirect?zip=60605"`;
my @ini = &read_ini( $service );
chop( @ini );
my $section = "";
do { $section = &process_section( $section ) } while
$section;
#$response = `./url.pl
"http://cgi.cnn.com/cgi-bin/weather/redirect?zip=60605"`;
exit;
##########################################################
```

```
sub read_ini {
     my ( $service ) = @_;
     my @ini = ();
     # first, try to read file
     $0 =~ m|^(.*/)[^/]*|;
     $service = $1 . $service;
     if ( open( INI, "$service.ini" ) ) {
          @ini = ( <INI> );
          return @ini unless ( $DB_SRV );
          # update datatable
          my $file_time = time - int( ( -M "$service.ini" )
* 24 *
3600 );
#         print "time $file_time\n";
          my $dbh = new Sybase::CTlib $DB_USR, $DB_PWD,
$DB_SRV;
          unless ( $dbh ) {
               print STDERR "webget.pl: Cannot connect to
dataserver $DB_SRV:$DB_USR:$DB_PWD\n";
               return @ini;
          }
          my @row_refs = $dbh->ct_sql( "select lastUpdate
from
mcServices where service = '$service'", undef, 1 );
          if ( $dbh->{ RC } == CS_FAIL ) {
               print STDERR "webget.pl: DB select from
mcServices
failed\n";
               return @ini;
          }
          unless ( defined @row_refs ) {
               # have to insert
               my ( @ini_escaped ) = map {
                    ( my $x = $_ ) =~ s/\'/\'\'/g;
                    $x;
               } @ini;
               $dbh->ct_sql( "insert mcServices values(
'$service',
'@ini_escaped', $file_time )" );
               if ( $dbh->{ RC } == CS_FAIL ) {
                    print STDERR "webget.pl: DB insert to
mcServices failed\n";
               }
               return @ini;
          }
#         print "time $file_time:".$row_refs[ 0 ]->{
'lastUpdate'
}."\n";
          if ( $file_time > $row_refs[ 0 ]->{ 'lastUpdate'
} ) {
               # have to update
```

```
                my ( @ini_escaped ) = map {
                    ( my $x = $_ ) =~ s/\'/\'\'/g;
                    $x;
                } @ini;
                $dbh->ct_sql( "update mcServices set config
=
'@ini_escaped', lastUpdate = $file_time where service =
'$service'" );
                if ( $dbh->{ RC } == CS_FAIL ) {
                    print STDERR "webget.pl: DB update to
mcServices failed\n";
                }
            }
            return @ini;
        }
        else {
            print STDERR "$0: WARNING: $service.ini n/a in "
. `pwd`
                . "Try to read DB\n";
        }
        # then try to read datatable
        die "webget.pl: Unable to find service $service\n"
unless ( $DB_SRV
);
        my $dbh = new Sybase::CTlib $DB_USR, $DB_PWD, $DB_SRV;
        die "webget.pl: Cannot connect to dataserver
$DB_SRV:$DB_USR:$DB_PWD\n" unless ( $dbh );
        my @row_refs = $dbh->ct_sql( "select config from
mcServices where
service = '$service'", undef, 1 );
        die "webget.pl: DB select from mcServices failed\n" if
$dbh->{ RC }
== CS_FAIL;
        die "webget.pl: Unable to find service $service\n"
unless ( defined
@row_refs );
        $row_refs[ 0 ]->{ 'config' } =~ s/\n /\n\r/g;
        @ini = split( /\r/, $row_refs[ 0 ]->{ 'config' } );
        return @ini;
}
#####################################################
sub process_section {
        my ( $prev_section ) = @_;
        my ( $section, $output, $content );
        my %Param;
        my %Content;
#       print"#################################\n";
        foreach ( @ini ) {
#           print;
#           chop;
            s/\s+$//;
```

```
            s/^\s+//;
            # get section name
            if ( /^\[(.*)\]/ ) {
#               print "$_: $section:$prev_section\n";
                last if $section;
                next if $1 eq "print";
#               next if $prev_section ne "" and
$prev_section ne $1;
                if ( $prev_section eq $1 ) {
                    $prev_section = "";
                    next;
                }
                $section = $1;
            }
            # get parameters
            push( @{ $Param{ $1 } }, $2 ) if $section and
/([^=]+)=(.*)/;
        }
#       print"++++++++++++++++++++++++++++++++++\n";
        return 0 unless $section;
#       print "section $section\n";
        # substitute parameters with values
        map { $Param{ URL }->[ 0 ] =~ s/$Param{ Input }->[ $_
]/$ARGV[ $_
]/g
        } 0 .. $#{ $Param{ Input } };

        # get page content
        ( $Content{ 'TIME' }, $content ) = &get_url_content(
${ $Param{ URL
} }[ 0 ] );

        # filter it
        map {
            if ( /\"([^\"]+)\"([^\"]*)\"/ or
/\/([^\/]+)\/([^\/]*)\// )
{
                my $out = $2; $content =~ s/$1/$out/g;
            }
        } @{ $Param{ "Pre-filter" } };

#print STDERR $content;
        # do main regular expression
        unless( @values = $content =~ /${ $Param{
Regular_expression } }[ 0
]/ ) {
            &die_hard( ${ $Param{ Regular_expression } }[ 0
], $content
);
            return $section;
        }
```

```
        %Content = map { ( $Param{ Output }->[ $_ ], $values[
    $_ ] )
        } 0 .. $#{ $Param{ Output } };

        # filter it
        map {
             if ( /([^\"]+)\"([^\"]+)\"([^\"]*)\"/
                  or /([^\/]+)\/([^\/]+)\/([^\/]*)\// ) {
                   my $out = $3;
                  $Content{ $1 } =~ s/$2/$out/g;
             }
        } @{ $Param{ "Post-filter" } };

        # calculate it
        map {
             if ( /([^=]+)=(.*)/ ) {
                  my $eval = $2;
                  map { $eval =~ s/$_/$Content{ $_ }/g
                   } keys %Content;
                $Content{ $1 } = eval( $eval );
             }

        } @{ $Param{ Calculate } };

        # read section [print]
       foreach $i ( 0 .. $#ini ) {
             next unless $ini[ $i ] =~ /^\[print\]/;
             foreach ( $i + 1 ..  $#ini ) {
                last if $ini[ $_ ] =~ /^\[.+\]/;
               $output .= $ini[ $_ ] . "\n";
             }
             last;
        }

        # prepare output
        map { $output =~ s/$_/$Content{ $_ }/g
        } keys %Content;
        print $output;
        return 0;
    }
    ##########################################################
    sub get_url_content {
         my ( $url ) = @_;
         print STDERR $url if $debug;
    #    $response = `./url.pl '$url'`;
         $response = `./url.pl '$url'`;
         return( $time - time, $response );
         my $ua = LWP::UserAgent->new;
         $ua->agent( 'Mozilla/4.0 [en] (X11; I; FreeBSD 2.2.8-
    STABLE i386)'
```

```
);
#     $ua->proxy( ['http', 'https'],
'http://proxy.webley:3128/' );
#     $ua->no_proxy( 'webley', 'vail' );
      my $cookie = HTTP::Cookies->new;
      $ua->cookie_jar( $cookie );
      $url = url $url;
      print "$url\n" if $debug2;
      my $time = time;
      my $res = $ua->request( GET $url );
      print "Response: " . ( time - $time ) . "sec\n" if
$debug2;
      return( $time - time, $res->content );
}
############################################################
sub die_hard {
      my( $re, $content ) = @_;
      my ( $re_end, $pattern );
      while( $content !~ /$re/ ) {
            if ( $re =~ s/(\([^\(\)]+\)[^\(\)]*$)// ) {
                  $re_end = $1 . $re_end;
            }
            else {
                  $re_end = $re;
                  last;
            }
      }
      $content =~ /$re/;
      print STDERR "The regular expression did not match:\n
$re\n
Possible misuse:
$re_end:\n
Matched:
$&\n
Mismatched:
$'\n
" if $debug;
            if ( $debug ) {
                  print STDERR "Content:\n $content\n" unless
$';
            }
}
############################################################
```

Once the web browsing server 302 accesses the web site specified in the URL 404 and retrieves the requested information, it is forwarded to the media server 304. The media server uses the speech synthesis engine 502 to create an audio message that is then transmitted to the user's voice enabled device 306. In the preferred embodiment, each

web browsing server is based upon Intel's Dual Pentium III 730 MHz microprocessor system.

Referring to FIG. 3, the operation of the personal voice-based information retrieval system will be described. A user establishes a connection between his voice enabled device 306 and a media server 304 of the voice browsing system 108 This may be done using the Public Switched Telephone Network (PSTN) 308 by calling a telephone number associated with the voice browsing system 108. Once the connection is established, the media server 304 initiates an interactive voice response (IVR) application. The IVR application plays audio message to the user presenting a list of options, which includes "perform a user-defined search." The user selects the option to perform a user-defined search by speaking the name of the option into the voice enabled device 306

The media server 304 then accesses the database 300 and retrieves the personal recognition grammars 402. Using the speech synthesis engine 502, the media server 304 then asks the user, "Which of the following user-defined searches would you like to perform" and reads to the user the identification name, provided by the recognition grammar 402, of each user-defined search. The user selects the desired search by speaking the appropriate speech command or pronounceable name described within the recognition grammar 402. These speech recognition grammars 402 define the speech commands or pronounceable names spoken by a user in order to perform a user-defined search. If the user has a multitude of user-defined searches, he may speak the command or pronounceable name described in the recognition grammar 402 associated with the desired search at anytime without waiting for the media server 304 to list all available user-defined searches. This feature is commonly referred to as a "barge-in" feature.

The media server 304 uses the speech recognition engine 500 to interpret the speech commands received from the user. Based upon these commands, the media server 304 retrieves the appropriate user-defined web site record 400 from the database 300. This record is then transmitted to a web browsing server 302. A firewall 310 may be provided that separates the web browsing server 302 from the database 300 and media server 304. The firewall provides protection to the media server and database by preventing unauthorized access in the event the firewall 312 for the web browsing server fails or is compromised. Any type of firewall protection technique commonly known to one skilled in the art could be used, including packet filter, proxy server, application gateway, or circuit-level gateway techniques.

The web browsing server 302 accesses the web site 106 specified by the URL 404 in the user-defined web site record 400 and retrieves the user-defined information from that site using the content extraction agent and specified content descriptor file specified in the content extraction agent command 406 Since the web browsing server 302 uses the URL and retrieves new information from the Internet each time a request is made, the requested information is always updated.

The content information received from the responding web site 106 is then processed by the web browsing server 302 according to the associated content descriptor file This processed response is then transmitted to the media server 304 for conversion into audio messages using either the speech synthesis engine 502 or selecting among a database of prerecorded voice responses contained within the database 300.

It should be noted that the web sites accessible by the personal information retrieval system and voice browser of the preferred embodiment may use any type of mark-up language, including Extensible Markup Language (XML), Wireless Markup Language (WML), Handheld Device Markup Language (HDML), Hyper Text Markup Language (HTML), or any variation of these languages.

The descriptions of the preferred embodiments described above are set forth for illustrative purposes and are not intended to limit the present invention in any manner Equivalent approaches are intended to be included within the scope of the present invention. While the present invention has been described with reference to the particular embodiments illustrated, those skilled in the art will recognize that many changes and variations may be made thereto without departing from the spirit and scope of the present invention. These embodiments and obvious variations thereof are contemplated as falling within the scope and spirit of the claimed invention.